OOPS 1 : Introduction

AGENDA

- → Programming Paradigm
- → Procedural programming
- → Object oriented programming
- → Access modifiers

PROGRAMMING PARADIGMS

Q what is a programming paradigm ?

↳ Style or Standard way of programming

Eg:- Way of writing code. For Example:- Burger can be made in many way but Each brand following defined structure so that the taste is always same.

=> without programming paradigms what will happen

(1) Less Structured

(2) Hard to read & Understand

(3) Hard to test

(4) Difficult to maintain etc.

Different types of programming paradigm

(1) Imperative Programming :-

It tells the computer how to do the task by giving a set of instructions in a particular order i.e. line by line.

Ex:-

```
int a = 10;
int b = 20;
int c = a+b;
print (c);
int d = b-a;
print (d);
```

→ Function

(2) Procedural Programming :-

It splits the entire program into small procedures or functions (section of code that perform a specific task) which are reusable code blocks.

```
int a = 10;
int b = 20;

addTwoNumbers (a, b);
SubtractTwoNumbers (a,b);

void addTwoNumber (a,b) {
    int sum = a+b;
    print (sum);
}

void SubtractTwoNumbers (a, b) {
    int dif = a-b;
    print (dif);
}
```

(3) Object oriented Programming

↳ It build the entire program using classes & object.

(4) Declarative Programming

In this paradigm, you specify "what" you want the program to do without specifying "how" it should be done.

Eg:- select * from Customer;
[SQL query]

QQ Based on Current programming we are doing which paradigm we are using?

↳ Procedural programming.

PROCEDURAL PROGRAMMING

↳ It splits the entire program into small procedures or function [section of code that perform a specific task] which are reusable code blocks.

Procedure —old name→ Function

QQ Execution of program starts from a specific procedure, what is it?

↳ main Function ()

For eg:-

```
void addTwoNumbers(a, b){
    int sum = a+b;
    print(sum);
}

void addThreeNumbers(a, b, c){
    int sum = a+b;
    addTwoNumbers(sum, c);
}

void main(){
    addThreeNumbers(10, 20, 30);
}
```

↳ Problems with Procedural Programming

Q. what you are doing right now?

↳ We are attending Lecture.

↳ I am having dinner.

↳ Chamandeep is teaching.

Generically

→ Subject + Verb (Someone is doing something)

i.e. entities perform action

Eg:-

```
PrintStudent(String name, int age, String gender){
    Print(name)
    Print(age)
    Print(gender)
}
```

OOPs So is there any way in procedural programming to combine set of attributes?

↳ yes using Struct or class

Eg:-

```
Public class Student {
    String name;
    int age;
    string gende;
}
```

Something ← Someone

```
Print Student ( Student st) {
    print (st. name);
    print (st. age);
    print (st. gender);
}
```

So, Something is happening on Someone.

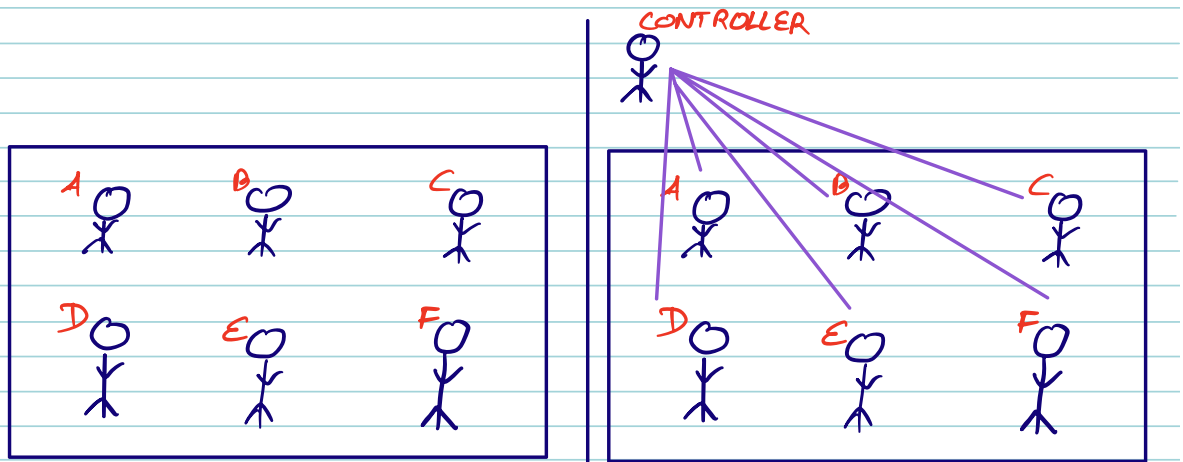
↳ Problem with Procedural programming

→ Its unnatural, because in real life entities perform action, but in procedural programming language action is performed on entity.

Procedure	OOPs
→ Procedure (Student)	→ Student . Print ( );
↳ Something is being done on someone	↳ Someone is doing Something.

★ Privacy is a Major challenge in Procedural programming.

⇒ Real Life Example



Object oriented word

↳ Here each individual have a control in hand.

Procedural Programming world.

↳ Controller controls everyone. [PUPPET SYSTEM]

↳ Controller can Manipulate the entities

Problem solved by

⇒ OOPs Example

Eg:-

```
class Student {
    String name;
    int age;
    String gender;

    Void print() {
        Print(name);
        Print(age);
    }
}
```

```
Student st = new Student();
st.print();
```

⇒ Cons of Procedural Programming

→ Difficult to make Sense.
→ Difficult to Debug.
→ Spaghetti of code i.e unstructured

OBJECT ORIENTED PROGRAMMING [OOPS]

→ Entities are core in OOPs

→ Every Entity have some attributes & behaviour.

↳ In OOPs using class & objects we will build the program (entity).

⇒ CLASS

→ Blueprint of an Idea.

Ex:- Floor plan of building

→ class represents structure of Idea.

Eg:-
```
class Student {
    int age;
    string name;
    double PSP;
    Pause Course () {==};
    change batch () {==};
    give Mock Interview () {==};
}
```

→ class takes no space in memory

→ Not a real entity

→ multiple instances of same class

⇒ OBJECTS

↳ The Real Instance of class

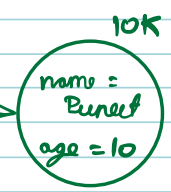
Quiz 1 :- Will object of a class occupy memory?

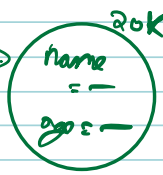
↳ yes

⇒ Creation of class & objects.

```
class Student {
    int age;
    String name;
    String batchName;
    void changeBatch(String newBatch){
        batchName = newBatch;
    }
    void giveMockInterviews(){
        SOP("Giving Mock Interview");
    }
}

// Object Creation.
Public Static void main(String[] args){
    Student st1 = new Student();
    st1.name = "Puneet";
    st1.age = 10;
    Student st2 = new Student();
    st2.name = "Vikas Yadav";
    st2.age = 20
```





PILLARS OF OOPS

→ 3 Pillar :- Support to hold thing together

→ 1 Principle :- Fundamental Foundation / Concepts

Ex :- Principle :- I will be a good person

Pillars →
(1) I will be truthful
(2) I will do hardwork
(3) I respect everyone

Similarly In OOPS

Principle of OOPs → ABSTRACTION

Pillars → ways to implement Abstraction
(1) Inheritance
(2) Polymorphism
(3) Encapsulation

SOURCE :- Java : The complete Reference

ABSTRACTION

↳ Hiding or Privacy? [Not Exactly]

↳ Representing in terms of Idea

Ex:- SCALER

↳ Students
Mentors
TAs

Q Do they have behaviours?

↳ yes, all of them have behaviours. like Student can take course, pause it

So, Abstraction is an idea of representing complex Software system interms of Ideas, because ideas are easy to understand.

⇒ Purpose of Abstraction?

↳ others dont need to know details of data.

ex:- ① Dynamic Array

② Taking a Left turn.

Q what needed to represented in terms of ideas?

↳ Data

↳ Anything that has behaviour.

PILLARS

① ENCAPSULATION :-

Q → Does the word Encapsulation contains the word which you are aware of?

↳ Capsule [Hold things together]

Q → what we really store in Programming?

↳ Attributes & behaviours

Quiz 2 :- where do we store attributes & behaviour together? what is the technical term for that?

↳ CLASS

② ACCESS MODIFIERS

We got to know that Encapsulation have two Advantages.

① One is to hold data & attributes together

② Second is it protect members from illegitimate access. you can't access the data from class unless allow you to.

↳ First point can be done using class.

↳ Second point can be implemented using **Access Modifiers**.

Quiz :- which one of these is not an access Modifier?

↳ OPEN.

⇒ Types of Access Modifiers :-

① Public

② Private

③ Protected

④ Default

Eg:-

```
class Student {
    Public int age;
    private String name;
    protected double PSP;
    String batchname;
}
```

① Public access modifiers

↳ A public attribute or method can be accessed by everyone.

② Private Access Modifier

↳ A private attribute or method can be accessed by no one, not even the child class.

Q :- If no one can access then what its purpose?

↳ Privacy / Security etc. .. ..

③ Protected Access Modifier

↳ Protected methods or Attributes can be accessed only from of the same package. And subclass of different package.

Package A

```
class XYZ {
    Protected int age;
}
```

Package B

```
class Student extends XYZ{
    // will be using age
}
```

④ Default access modifies → within same package.

	class	package	subclass (same pkg)	subclass (Diff pkg)	world
Public	✓	✓	✓	✓	✓
Protected	✓	✓	✓	✓	
Default	✓	✓	✓		
Private	✓				

Quiz 4 :- which is most Restricted Access Modifier

↳ Private.

Quiz 5 :- which is most open Access Modifier ?

↳ Public

THIS KEYWORD

↳ Used to refer Current Instance of class or Object.

↳ Helps in Differentiating instance variable & local variable or Method parameter with same name, & to access or modify instance members within Method.

Eg:-

```
class Person {
    String name;
    void changeName(String name) {
        this.name = name;
    }
}
```

```
Person P = new Person();   // 10k
P.changeName("Vikas");

Person P2 = new Person();  // 20k
P.changeName("Puneet");
```

10k: name = Vikas

20k: name = Puneet

STATIC KEYWORD

↳ In Programming languages like Java & C++ is used to Declare class-level members or methods.

↑ which is associated with class itself

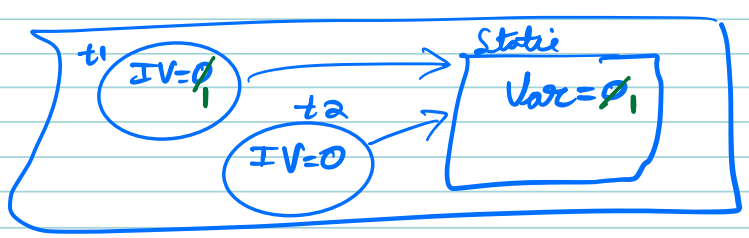
① Static Variable

↳ when you declare a variable as "static" within a class, it becomes a class variable. These variables are shared among all instances of the class. They are initialized only once when the class is loaded, & their values are common to all objects of the class

Ex:-

```
Public class MyClass {
    static int Var = 0;
    int instanceVariable;          → [IV]
    Public MyClass() {             ← Constructor
        instanceVariable = 0;
    }
    Public void Test() {
        instanceVariable++; Var++;
    }
}
```

Heap



```
MyClass t1 = new MyClass();
Print (t1.instanceVariable);   ⟶ 0
Print (t1.Var);                ⟶ 0
t1.Test();
Print (t1.Var);                ⟶ 1
MyClass t2 = new MyClass();
Print (t2.instanceVariable);   ⟶ 0
Print (t2.Var);                ⟶ 1
Print (MyClass.Var);           ⟶ 1
```

Observations

1. Static variable is created once for all classes.
2. Static variable can be accessed directly from class Non.

(2) Static method (class methods)

↳ when you declare a method as "static" within a class, it becomes a class method. These methods are invoked on the class itself, not on instances of class. They can access static variables & perform operations that doesn't require access to instance-specific data.

Eg:Eg:-

```
Public Test {
    Public static void main() { }
    Public static void swap() { }
}
Test.swap();
```

Scope of a Variable

① Class / Static Scope

↳ Variables declared as static within a class have class level scope. They are associated with class rather than instance, can be accessed with class name & shared among all instances.

② Instance Scope

↳ Variable defined in class but outside any method / constructor have instance level scope. Associated with a instance of class & each instance has its own copy of these variables.

③ Method / Local Scope :- Variables declared within a method of class. Can be accessed within specific method.

④ Block Scope :-

Variables defined within { } [Curly braces]. can be accessed within same { }.

```
class Dummy {
    Static int age = 10;        ⟶ Static Level
    int name = "Charandeep";    ⟶ Instance Level
    Print Counting () {
        int start = 1;          ⟶ Method level.
        for (i=0; i<start; i++){
            int temp = i+1;     ⟶ Block Level
            SOP (temp);
```